Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Liisa Sailaranta

Real-time predictions in Web services

Master's Thesis
Espoo, January 14, 2017

Supervisor: Associate Professor Keijo Heljanko, Aalto University
Advisors: Timo Lehtonen M.Sc. (Tech.)
Teppo E. Ahonen Ph.D

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Liisa Sailaranta

Title:
Real-time predictions in Web services

Date: January 14, 2017 **Pages:** 62

Major: Computer Science **Code:** SCI3042

Supervisor: Associate Professor Keijo Heljanko

Advisors: Timo Lehtonen M.Sc. (Tech.)
Teppo E. Ahonen Ph.D

In this Master's Theses a real-time analytics pipeline is built to serve predictions to users based on the usage and the operational data of a Web service. The data of the service is analyzed and a predictive model is built using statistical learning methods. The pipeline is set up to serve the predictions real-time using components from Amazon Cloud Services.

The aim is to show the user a prediction of how long will it take until she/he gets a verdict on her/his application from the service. As additional goals, the aim is to study the dataset and its possibilities and research the suitability of the Amazon Machine Learning service in real-time predictions in Web context.

The features for the predictive model are selected by exploring the dataset and using the Amazon Machine Learning service to evaluate the features. The Amazon Machine Learning service is also used to build a predictive machine learning model. The real-time analytics pipeline is built using Amazon components and following the Lambda Architecture guidelines.

The best model performed better than the baseline model, though only moderately. The data lacked some vital information for the prediction target such as information about the personnel. Implementing the pipeline with Amazon components was considered straightforward. The Lambda Architecture worked well for the problem. It was found out that the Amazon Machine Learning service is easy to use but its machine learning capabilities and user interface are limited. It was highlighted that it is essential to explore and learn the dataset before building or designing the pipeline, as the pipeline design depends heavily from the data and from the use case.

Keywords: Machine learning, Real-time analytics, Statistical learning, Cloud computing, Lambda Architecture

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Liisa Sailaranta		
Työn nimi:	Reaaliaikaiset ennustukset verkkopalveluissa		
Päiväys:	14. Tammikuuta 2017	**Sivumäärä:**	62
Pääaine:	Tietotekniikka	**Koodi:**	SCI3042
Valvoja:	Professori Keijo Heljanko		
Ohjaajat:	Diplomi-insinööri Timo Lehtonen Filosofian tohtori Teppo E. Ahonen		

Tässä diplomityössä on rakennettu reaaliaikainen analytiikkajärjestelmä, jolla näytetään ennustuksia käyttäjille eräässä verkkopalvelussa, perustuen verkkopalvelun käyttödataan ja operatiiviseen dataan. Verkkopalvelun dataa analysoidaan ja sen perusteella rakennetaan tilastollisiin menetelmiin pohjaava ennustava koneoppimismalli. Analytiikkajärjestelmä rakennetaan käyttäen komponentteja Amazonin pilvipalvelusta.

Tarkoituksena on näyttää käyttäjälle ennustus siitä kauanko kestää, että hän saa vastauksen verkkopalveluun jättämäänsä hakemukseen. Tämän lisäksi tavoitteena on muodostaa ymmärrys verkkopalvelun datasta ja sen mahdollisuuksista, sekä tutkia soveltuuko Amazonin koneoppimispalvelu reaaliaikaisten ennustuksien näyttämiseen verkkoympäristössä.

Ennustavan mallin ominaisuudet valittiin tarkastelemalla dataa ja evaluoimalla ominaisuudet Amazonin koneoppimispalvelun avulla. Amazonin koneoppimispalvelua käytettiin myös ennustavan koneoppimismallin rakentamiseen. Reaaliaikainen analytiikkajärjestelmä rakennettiin käyttäen komponentteja Amazonin pilvipalveluista ja seuraten Lambda-arkkitehtuurin suunnitteluperiaatteita.

Paras rakennetuista koneoppimismalleista oli parempi kuin pohjamalli, joskaan ei mitenkään merkittävästi. Datasta puuttui joitain ennustettavan arvon kannalta tärkeitä tekijöitä kuten tietoa hakemuksia käsittelevästä henkilökunnasta. Analytiikkajärjestelmän rakentaminen Amazoniin osoittautui kuitenkin helpoksi. Amazonin koneoppimispalvelu todettiin helppokäyttöiseksi, vaikkakin se todettiin koneoppimisominaisuuksiltaan melko yksinkertaiseksi, sekä käyttöliittymän osalta rajoittuneeksi. Työssä korostetaan, että on tärkeää tutkia dataa ennen kuin rakentaa analytiikkajärjestelmän, sillä järjestelmän rakenne riippuu suuresti siitä minkälaista data on ja mikä on sen sekä datan käyttötarkoitus.

Asiasanat:	koneoppiminen, reaaliaikainen analytiikka, tilastollinen oppiminen, pilvilaskenta, Lambda-arkkitehtuuri
Kieli:	Englanti

Acknowledgements

I would like to thank my supervisor Keijo Heljanko for support and guidance during this Master's Thesis.

I would like to thank my instructor Timo Lehtonen for this interesting topic for the Thesis and all the support and guidance during the process. I would like thank my instructor Teppo E. Ahonen for support on working with the text and for his great substance knowledge and professional attitude.

I would like to thank my colleague Jussi Määttä for inspiration and ideas on the data. I would like to thank Lea Hämäläinen for proofreading this Master's Thesis. I would like thank all my colleagues and friends who read and commented my Thesis and gave me new ideas and encouraged me in the process.

Finally I would like to thank my family for all the support during my studies and during this Master's thesis.

Espoo, January 14, 2017

Liisa Sailaranta

Abbreviations and Acronyms

AML	Amazon Machine Learning service
CSV	Comma Separated Values
API	Application Programming Interface
REST	Representational State Transfer
RMSE	Root Mean Square Error

Contents

Chapter 1

Introduction

Machine learning has many interesting applications in Web services. Web services can be optimized and customized with the help of machine learning methods. A Web service can automatically recommend content to a user based on his previous behavior on the Web service. Statistical learning models can give the user insight to his process and give him additional information about his case.

There are however challenges on the way of making machine learning part of every day Web development. Many of the machine learning applications and data analysis technologies lack scalability and robustness. On the other hand most of software developers lack the theoretical knowledge of machine learning to implement predictive machine learning applications. Big amounts of data and demand for real time data processing also set demands on software developers and architects.

In this Master's Thesis we are going to build a real time analytics pipeline to bring real time predictions in a Web service. The pipeline will show a real-time prediction to the user visiting the Web service. The prediction will give user insight to his process and add transparency to the Web service. The predictions will be based on the usage and operational data of the Web service.

By building this pipeline we learn about the challenges of bringing machine learning to a Web service and the motives and possibilities of machine learning in Web context. To simplify our pipeline we will be using the Amazon Machine Learning service to provide the machine learning functionality to our pipeline. The Amazon Machine Learning service is a software as a service solution built by Amazon to provide an easy to adapt machine learning service to every day software development. We will evaluate the Amazon Machine Learning service and its suitability to solve our problem.

The service we are going to adapt our real time analytics pipeline to, is Lupapiste Web service, which is a service for applying for a construction permit

from a municipality. The service was built by Solita[1]. It has extensive logging data from the usage of the service over the service lifetime. Our pipeline aims to predict information from the log and operational data sets of the service that would be beneficial for the user and to the authorities. By bringing the machine learning dimension to the Lupapiste service, we wish to bring more transparency to the service and learn about the Lupapiste data set and its possibilities.

1.1 Problem statement

In this Master's Thesis we will research how well the Lupapiste data set and the Amazon Machine Learning service answer our goal of creating real-time predictions in a Web service and what is the pipeline architecture that suits our needs. We will analyze our data and discuss how well the data answers to our problem and what sort of preprocessing is needed for the data. We will discuss the architecture of the real time analytics pipeline and implement the essential parts of it.

We will explore the Lupapiste data and build a predictive machine learning model based on the data using the Amazon Machine Learning service. We will choose the features for our machine learning model based on the analyses of the business logic of the Lupapiste service and explorative analyses on the data. The Amazon Machine Learning service provides us the functionality to evaluate our assumptions on the data and train and build the machine learning model. Once the machine learning model is built and running in the Amazon Machine Learning service, we will design and build a real time analytics pipeline using Amazon components. The aim of the real time pipeline is to bring the prediction to the users in real-time based on the model we built. The pipeline will adapt the architecture of the Lambda Architecture.

The first part of our problem is to find out whether our data can predict the desired target and how accurate this prediction is. Our data set will be a combination of Lupapiste usage data logs and Lupapiste operational data from the production database. Based on that data we want to show to the user a prediction of how long his application processing time will be. By processing time we refer to time that goes after user submits the application to the point the municipality gives a verdict on the application. We will explore the data to find the features that have correlation with our target that is the processing time. Once we have good feature candidates we will build a machine learning model from the candidates using the Amazon Machine

[1]Solita is a Finnish software and consulting company founded on 1996. It develops digital solutions to customers in private and public sectors.

Learning service and let the service evaluate our model. Based on the evaluation we know whether our feature candidates actually have any correlation with the target and whether it improves our model. Using the Amazon Machine Learning service we can iterate over all of our feature candidates to find the best predicting model.

The second part of our problem will be the integration of the Amazon Machine Learning service with the Lupapiste service to bring the real time predictions to users. The Amazon Machine Learning service is a ready built stand-alone Web service, so no implementation or infrastructure set up is needed there. The service also offers an API for requesting the real time predictions. Therefore the main problem we will need to solve in this phase, is how to gather all the needed information for the prediction and how to serve the prediction real time to the user. To solve these problems, we will set up the Lambda Architecture type pipeline in a cloud environment containing several databases and other components. We will analyze the scalability of our pipeline and discuss the price and information security perspectives. We will discuss about the value cloud computing can bring to data analysis and machine learning solutions.

For all our infrastructure we choose Amazon cloud services due to Amazon being a big, popular cloud service provider and widely used at Solita. There are similar solutions and even similar machine learning solution to the Amazon Machine Learning service offered by Microsoft Azure and Google Cloud, but we will concentrate on Amazon solutions in this theses. In some extend our solution could be applied to other providers, but this will be left for further research.

1.2 Structure of the thesis

In Chapter 1 we introduce our problem and the scope and describe the structure of this Master's Theses.

In Chapter 2 we will go through the machine learning basic principles using fundamental literature of this area. We will take a look of previous research made in the past on machine learning and real-time prediction in Web services. We will shortly discuss cloud services and introduce the Amazon Machine Learning service. At last we will introduce the Lupapiste service.

In Chapter 3 we will introduce an approach called linear regression used in the model building. We will explain the method for selecting the features. We will introduce a design called Lambda Architecture, that we will use for our pipeline.

In Chapter 4 we will run the data analysis, discuss and present the results

of the analysis. We will describe the process needed for preprocessing and feature engineering the data and describe the process of building the model with Amazon Machine Learning service. We will describe the implementation of our pipeline.

In Chapter 5 we will discuss our model and possible ways to improve it. We will discuss the scalability and other attributes of our suggested pipeline.

In Chapter 6 we will present our conclusions, by summing up what we learned from building our machine learning model and data pipeline.

Chapter 2

Background

2.1 Machine learning in Web services

In this Master's Thesis we want to bring real-time predictions to users in our Web service. To get background to our problem we will first take a look at similar research done in the past related to real-time predictions and machine learning in Web services.

Real-time predictions in Web services has been a research topic already already from the early days of the Web [11]. The interest for this problem setup comes from the desire to recommend user content in a Web service real-time. The motive behind this or any other personalization or customization based on user's profile is to engage users to the Web service or increase the conversion rate of the Web service [25]. Especially in the Web services such as e-commerce Web services, where conversions bring direct profit, different sorts of recommendation systems have become more a necessity than an additional feature. A system which can profile users and recommend content based on the profiling can be referred to a *recommendation engine*.

The first real-time prediction systems were based on preferences and information a user inputted himself in a Web service [22]. User was shown for example questionnaires on a Website or was offered a possibility to rate the content while using the service. This however resulted to sluggish user experience and collected possible biased data as users failed to judge their interests and motives objectively [22]. Questionnaires and forms in Web services also raised information privacy concerns and users were not always willing to offer personal data about themselves [11].

To avoid the concerns of user inputted data and also to find more unexpected patterns and connections in data, the existing data like click stream data, application logs or visited URLs can be mined for finding the user pro-

files or patterns [23][26][22]. The traditional approach for creating predictions from this sort of usage data is comparing the new user to history records using clustering methods and selecting the users with most similar behavior in the Web service [26]. For example we can examine the URLs user visited on Web service and group users by those URLs. That would mean if users A and B visited URLs a and b and user C visited URLs c and d, we would divide our users to two clusters, one with A and B and other with the user C. If we now have a new user D which would have visited URL c we would divide him to same group with C since they have most same visited URLs. Now based on other content user C has visited, we could recommend to user D the content d. This approach can be called *collaborative filtering* [17]. Despite being a simple approach this is a relative efficient and intuitive solution for this problem. It is used by big, modern Web services such as Amazon e-commerce Web service and YouTube Web service [20][8].

In reality clustering is not this straightforward. Especially in the Web services of today, where user can visit hundreds of URLs during one session, defining the distance between two users' profile can get complicated. The pure clustering approach also lacks scalability: finding the right cluster for new entry might end up in a unbearable latency with multidimensional data [23]. Therefore lots of research has been done to define and optimize the process of calculating the distance between two users profile. For example in case of URLs, different URLs in the Web service can be given different weight when calculating the distance to emphasize the essential content on the Web service [26]. Also instead of clustering the visited URLs we can examine the sequence of the URLs and recognize patterns in the sequences the user visited [22][4]. In addition to URLs, in context of e-commerce for example the previous purchases or in context of social media the relationships between the users or user's search queries can be used as attributes when building the clusters [17][14].

In the earlier research a clear separation is made between offline and online processes of the predictive machine learning workflow [22]. By offline process is referenced to the phase were the data is pre-processed and the clusters are formed. By online process is referenced to the phase where user is offered the prediction in the Web service. The modern Web services of today need to however handle huge amount of rapidly evolving data and several different variables when building the predictions. Therefore the separation between offline and online process is no longer an working analogy but services have complicated data pipelines specifically built for collecting, processing and serving the data in real time [29][19]. For these purposes specific Big Data solutions have been developed for fast and robust data access, process and storage [12]. Most notable of these technologies are collection of database

technologies called NoSQL databases and batch and stream processing frameworks for efficient data processing [6][9].

The data pipelines essentially collect and aggregate data from different data sources for example from different log files or real-time data streams. After that the pipeline usually processes some calculations over the data and stores the processed results to separate storage for further analyzes or for serving results to users. The pipelines need to deal with heavy write operations, evolving data schemas and offer high availability, fault isolation, real time view on the processed data [5][29][7]. To meet these requirements different components in the data pipeline might for example need be duplicated and many completely new architectural designs need to be made.

The problem we are trying to solve in this thesis differs slightly from the earlier research presented here. Firstly, our motive differs from the most typical use case of real time predictions. We are not trying to engage the user on our Web service or recommend him content. We want to give user information based on our model. Our motive is to bring transparency to the process and by that in the long run to accumulate the process. Secondly, our case differs from the cases described here since instead of blindly clustering our data we have some prior knowledge of our users. For users who have already gotten a decision to their application in the service, we already know our target which is the processing time of the application. We can utilize this information when creating the predictions for the new users. The approach we use in this thesis can be called *supervised learning*.

We will open the term supervised learning and the statistical learning concepts powering the predictions in Chapter 2.2.

2.2 Statistical machine learning

To understand the method of creating predictions based on existing data we will shortly go through the principles and basic methods of statistical learning. We will build here the necessary theory of statistical learning needed to follow the rest of the thesis.

In statistical learning we try solve the following problem: if we are provided with variables X how can we predict the output value Y based on those variables. To open this we go through a simple example: sales predicting. In this example we are selling a product and we are interested in the sales of it. Therefore the sales is the output value Y that we are trying to predict. The amount of sales depends on several variables such as TV advertising, social media advertising or re-sellers. These variables are described as X. Essentially the problem we are trying to solve is to find a function f to fit our

variables X to our target Y [15]. In other words we try figure out how do different variables X (TV advertising, social media advertising or re-sellers) effect our target Y (sales). After finding the f, we can use the f to predict target Y based on the variables X for completely new entries. That means we could predict or at least give an estimation of the sales based on planned advertising and re-sellers already beforehand or even better we can select our TV advertising, social media advertising and re-sellers to optimize the sales.

Essentially most of the statistical learning problems fall into two categories: supervised and unsupervised learning. The example of sales previously is an example of supervised learning. For each sample there is a corresponding target value Y that is *supervised output*. In contrast in unsupervised learning there is no supervised output available for the data [15]. To explain the idea of unsupervised learning, we will go through an other example. If we would be selling a product we would have an idea of the groups of people buying our product. Each group could be described by the motives and demographics of the group. Being able to fit a new potential buyer to any of our user groups would be beneficial, since we might be able to target our advertising and supply better. However these groups are not known beforehand and therefore no supervised method can be applied here. To find out these user groups, we must use unsupervised methods for clustering the data [15]. These clusters then represent our buyer groups. Once the clusters are known we can fit a new buyer to one of the clusters and use that to predict a user's motives or behavior. This problem of user groups is similar to the problem of grouping users by visited URLs described earlier.

In this thesis we are going to use the supervised learning method. Our problem is more similar to the first example of sales than to second example of the buyers. We have the variables X and also the supervised output Y to predict. Essentially supervised learning processes consists from two phases, the training and the valuation phase. In the training phase we will estimate the f based on the training data with supervised output. After training we go through the an evaluation phase for testing our model against evaluation data, which must be different from the training data [15]. This evaluation data set must also have supervised output Y in order to evaluate the model performance. If we are satisfied with evaluation results, we can use our model to give predictions for new data entries.

Developing a statistical learning model also includes a feature selection phase. That means we try to select the variables that give us the best model. In the rest of this Master's Thesis we will call the these variables *features*. We can improve our features by extracting more information out of the original data or phenomena. This sort of feature extraction requires a sufficient domain understanding [33]. There are also automatic methods for selecting the

significant features, but in this thesis we will not be using any of those but select our features based on domain understanding. Automatic methods for feature selection are not needed in our research since the variable set in our data is relatively small. Also we want to build a good understanding from the data and by selecting features through explorative data analysis we get to examine and learn about the data.

2.3 Amazon Machine Learning service

The Amazon Machine Learning service functionality is based on the statistical learning methods described before. We will discuss our motivations and interests for using the service in this Master's Thesis.

The Amazon Machine Learning service is a software as service solution developed by Amazon[1]. It was published 2015 to bring Amazon to the market field of low cost fast adapt machine learning solutions. The service offers both an user interface and APIs for building and evaluating a predictive machine learning model and fetching predictions based on the model. The service offers both real-time predictions and batch predictions for bigger amounts of data.

The details of the implementation of the Amazon Machine Learning service are not public, but we can come up with some conclusions of the implementation based on the documentation and the API of the service. Based on the documentation we can say that the machine learning functionality is based on linear regression and logistic regression algorithms. These algorithms are introduced in Chapter 3.1. Both of these algorithms are efficient algorithms suitable for many machine learning problems but also relatively basic and standards in the industry [24][32].

As there is nothing pivotal in the machine learning functionality of the Amazon Machine Learning service, to understand the motivation behind it we must examine it from a system architecture point of view. Based on previous discussion and introduction to machine learning methods it is easy to see that building a machine learning application is not trivial. It requires understanding about statistical methods, machine learning technologies and the data set itself. The application easily becomes heavily domain specific and difficult to test or write the exact specifications other than in the most trivial cases [28]. There are solutions described in literature to use batch processing frameworks for building machine learning applications [13][19]. That sort approach however requires knowledge over machine learning methods and also

[1]http://docs.aws.amazon.com/machine-learning/latest/dg/machinelearning-dg.pdf

deep knowledge about the batch processing framework implementation to be able to run algorithms efficiently. With Amazon Machine learning service the company or the team can take machine learning service as part of the system easier, without a need to build and maintain a complex machine learning system on their own. Even in a case where the company would be interested in investing in building a machine learning solution in the long run, with the Amazon Machine Learning service or with any other similar solution by any other service provider, the machine learning feature can be tested and prototyped with low cost.

Being a cloud solution the Amazon Machine Learning service also offers scalability that the common tools for analyzing data and building models such as R and Matlab lack by default [3][30]. Even though there are commercial and community solutions for bringing scalability to Matlab and R, those tools can not yet be trivially scaled to run on several machines [28]. Machine learning applications will by design need to be able to handle big data sets. Building a machine learning model is an iterative process which requires lots of computing power. However the need for high computation power is not constant since building the model is a temporary process and does not need to run constantly. The cloud services have the appearance of infinite computing power which is available on demand [2]. By using cloud services in machine learning, it is possible to avoid big upfront commitment and only pay for resources that are needed. Therefore we do not end up constantly paying for the computing power that is only needed for relatively small amount of time when building the model or creating predictions. This circumstance is also reflected in the Amazon Machine learning service billing policy. The usage of the service is billed by time when building the model. The predictions are billed, by the amount of predictions requested from the service.

2.4 Lupapiste service

The Web service that we are going to target in our research is the Lupapiste Web service[2]. We will next explain the basic functionality of the service and discuss our motivation of bringing a machine learning dimension to the Lupapiste service.

Lupapiste is a Web service for applying construction permission from municipalities. It was released in the spring 2013. By end of year 2016 it is in use approximately in 100 municipalities in Finland. The Lupapiste service simplifies the application process for both applicant and municipality authorities,

[2]www.lupapiste.fi

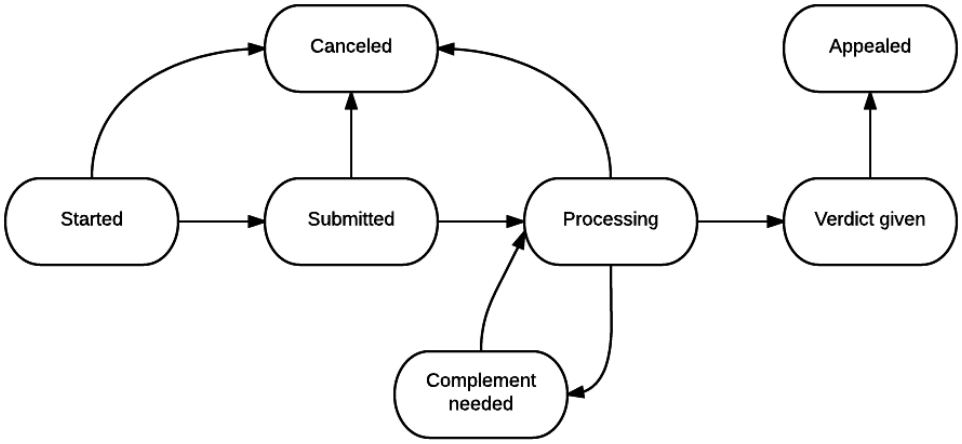

Figure 2.1: The state diagram of application in the Lupapiste service

by providing a fluent communication channel for all parties included in the construction process and simple, informative user interface.

To get the construction permission one must declare detailed a plan for the building project, the ownership, the materials and the techniques to be used in the construction project. The municipality authorities will make a verdict of a construction permission based on the offered information and documents. Especially for large projects, the key personnel such as lead designer or project manager must also be accepted by the municipality. The application process states are shown in the state diagram in Figure 2.1. In this diagram we see the Lupapiste process states from creating the application to the state where application is assigned to the municipality and finally given a verdict.

It is typical that the processing time takes longer than excepted. Additional information from the applicant is often needed, when all needed information is not submitted in the initial submission. The service does prevalidation of the data user entered such as mandatory attachments for saving time of the municipality officers in most trivial cases. However maintaining detailed validation rules for tens of available application types soon turns out to overly troublesome. Due applicant failing to give out all the needed information in the first place the application goes back to applicant to be complimented and from there again to be processed by authorities which results to extended processing time. Also other matters, like municipality personnel work situation effect the application processing times.

In this Master's Thesis, we are trying to model the application processing

time based on data available at the moment the user submits the application. By processing time we refer to the time that goes from submitting the application to the point the verdict is given by the municipality. We want to show a prediction of the processing time to the applicant real-time in the Lupapiste service after the applicant submits his application. Based on this information the applicant can estimate how long it is going to take to get the verdict. We chose the processing time as prediction target, since generally it is the single most interesting variable to the applicant. Predicting the processing time is also interesting, since the authorities want to shorten the processing time. Therefore they are interested to understand the variables that cause the application process to lengthen.

Chapter 3

Methods

3.1 Linear regression

We will build and evaluate our machine learning model using the Amazon Machine Learning service. For building the machine learning model the Amazon Machine Learning service uses an approach called *linear regression*. Understanding the linear regression algorithm is not mandatory for using the Amazon Machine Learning service as the service hides most of its internal functionality. We will however go through the idea of the linear regression, as this will later help us to understand and analyze our results.

Linear regression is based on the idea of finding a linear formula for predicting target Y based on variable X. It is a supervised learning approach. To use this algorithm there must be a linear relationship between the variable X and target Y [15]. Mathematically we can write this linear relationship as follows:

$$Y \approx \beta_0 + \beta_1 X, \tag{3.1}$$

$\approx$ meaning *approximately modeled as*. The β_0 and β_1 in Equation 3.1 are two unknown constants: *coefficients*. Once we manage to give estimations for the *coefficients* we can use this equation as a model to create prediction for target Y based on variable X. We can also present the idea of linear regression visually as in Figure 3.1. In the figure we use red dots to present our sample data and a blue line to estimate the linear relationship between the variables and target. The aim of linear regression is to fit the blue line so its best describes the data.

We evidently try to select our coefficients β_0 and β_1 so that our predictions are as close as possible to the supervised output values. The standard way for measuring this *closeness* is the minimizing *least squares* criterion [15]. From

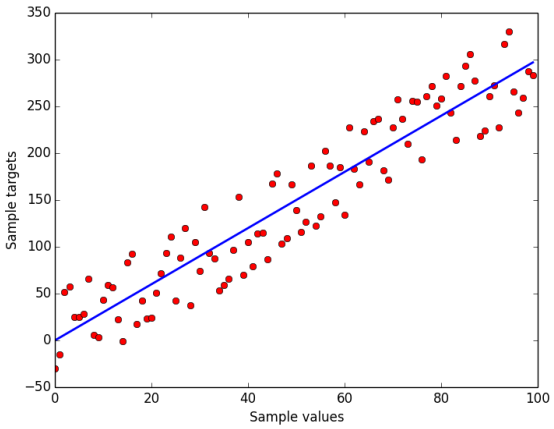

Figure 3.1: Fitting a linear regression line to sample data

this criterion we can lead a formula for calculating the coefficients. To understand this a little better we will go through the idea of the minimizing the *least squares* criterion. To calculate the *least squares* -error we must first calculate the predicted i:th value from the data as $\hat{y}_i = \beta_0 + \beta_1 x_i$, where β_0 and β_1 are our estimated coefficients. Based on that we can calculate the *residual*, the difference between our predicted i:th value and the i:th supervised value of the data as $e_i = \hat{y}_i - y_i$. Then we calculate the sum over the squares of all residuals. This is called as the Residual Sum of Squares (RSS) and presented formally as RSS $= e_1^2 + e_2^2 + ... + e_n^2$ [15]. By creating a formula to minimize this RSS, we can after some calculus come up with following formulas for coefficients:

$$\hat{b_1} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})} \tag{3.2}$$

$$\hat{b_0} = \overline{y} - \hat{b_1}\overline{x}, \tag{3.3}$$

where $\overline{y}$ is the mean of values Y and $\overline{x}$ the mean of values X. From Equations 3.2 and 3.3 we get the estimations for β_0 and β_1 and placing them to our original Equation 3.1 we can now create predictions for new data entries.

The case presented here is a simple form of linear regression. We only have one variable that the target depends on. In most of the cases, also in our research case, the target depends on several variables. The case where linear regression is applied to multiple parameters is called multiple linear regression [15]. Instead of having a single variable effecting the target Y, we have multiple variables with their coefficients:

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p \tag{3.4}$$

Unlike with simple linear regression with one variable, estimating the coefficients in multiple linear regression can not be represented as a simple formula so we will not present the formula here. Essentially the idea of estimating the coefficients is the same as with simple linear regression. The biggest challenge the multiple regression poses is finding the significant features from the variables. Different variables might perform differently when combined in the same model than when researched individually [15]. To tackle this when building the linear regression model, we will need to try different combinations of our features.

Linear regression requires the data be of a linear type, meaning that the distance between data points can be given a value. It only suits numeric values. The Amazon Machine Learning service however lets us also predict binary and categorical values. A method called logistic regression is used there. Our target will however be a numeric value so in this research we will use the linear regression.

Another limitation of the linear regression approach is that it only performs well with data, to which a linear model can be fitted naturally [15]. That means that if the variables X and the target Y do not have a linear relationship the linear regression can not fit a model with good predictions. In this case alternative methods such as Support Vector Machines could bring better results [15]. However the Amazon Machine Learning service does not support other methods than linear regression. We however assume that our data has approximately a linear correlation with the target. Therefore we assume that the linear regression model is a suitable approach for us.

3.2 Feature selection

We will start the model building by extracting the significant features from the data. The method for finding the significant features will be the following. We will first come up with ideas of the features by analyzing and understanding the business context of the Lupapiste service. Then we will examine the data by plotting figures to verify our assumptions.

When we have a good feature candidate, we will test our feature using the Amazon Machine Learning service linear regression algorithm. The Amazon Machine Learning service builds a linear regression model from our features and data we upload to the service. This model would be technically already as such ready to be used for predictions, but we will in this phase only use it for testing our features.

The service evaluates the model by calculating the Root Mean Square error (RMSE) to our values. RMSE is a similar value than RSS described in Chapter 3.1. In addition of summing the squares of the residuals the RMSE takes a mean of the squares and then the square root from the mean. We can present this mathematically as following:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=0}^{N} e_i^2}, \tag{3.5}$$

where N amount of the amount entries in our data. By using the RMSE we get the idea of the error per sample. Instead of looking at a big sum of errors, we examine the single error which we can compare to our data values. This way we can get an idea of how much the error actually is proportioned to the data samples. However it is good to notice that even though the RMSE is a single constant value, the error per sample is not constant among samples, but differs depending on the sample. The way RMSE is calculated emphases big errors. The samples that differ a lot from the other samples have a big error close to the RMSE, but the samples that fit well to the linear model might actually have smaller error than the RMSE suggests.

By using the RMSE we can analyze whether our feature brings any value for the model. If the RMSE seems to be bigger than with previous feature sets we should probably drop out this feature. If the RMSE however is smaller than with previous features, the feature improves our model and we should keep it. However as discussed Chapter 3.1, the same feature might show correlation to the target in some feature combinations but in some other combinations show no correlation. The feature that first seemed insignificant might seem significant when combined with some other features. Due this we should try out different feature combinations to find the most optimal model.

The benefit of our method is that we do constant cross checking of our assumptions about the data with the Amazon Machine Learning service. We do not build a separate model for evaluating the data, which might give us results that would not later be repeatable with Amazon Machine Learning service. We can constantly be sure that Amazon sees and understands the features in a similar manner that we do. Once we are done with feature engineering and selection we will have a ready to use machine learning model in Amazon with no need to start building a new model on another platform or going through the training and the validation processes again.

3.3 Lambda Architecture

To bring the real time predictions to the Lupapiste users we must build a real time analytics pipeline, that will deliver the predictions to the users real time. At high level the pipeline has following requirements. It should take in one data entry: the application, do the needed preprocessing to the application, request a prediction for it from the Amazon Machine Learning service and return the prediction to the Lupapiste service. The Lupapiste service will handle showing the prediction to the users in the user interface.

For requesting the prediction from the machine learning service we need to have some information about the application based on what the machine learning service can give the prediction. Earlier we dubbed this information *features*. What these features are in our case is discussed in Chapter 4.2. Our data is log events enriched with operational data. We can call this primitive data just purely *data*. To be able to turn data into features we must be able to derive our data into *information*. To turn the data into information we must be able to run *queries* over data. These queries aggregate information from data in different ways. They might calculate sums over data or calculate distinct values in a column. To not pose any limitations for queries, they must essentially be able to run over the whole data set. Formally we could define the query as following:

$$\text{query} = \text{function}(\text{all data}) \tag{3.6}$$

By running the query over the whole data set we will be able to answer questions about the data such as how many times did certain events occur in the data or what is the final status after each event. However running queries on all the data will in the long term require lots of resources and result into huge latency. Over the Lupapiste service lifetime the service is going to collect gigabytes of log data, that would all need to be analyzed in each query. There are solutions specifically designed for efficient big data batch processing. However even with these solutions the latency soon becomes unbearable for real-time predictions and also the required resources get really expensive.

To tackle this problem of fast queries on a big constantly growing data set a design called Lambda Architecture has been described [21]. The Lambda Architecture describes a system which provides implementing any arbitrary function over any data with results returning in low latency. Essentially the Lambda Architecture consists from three layers: a batch processing layer for big data batches, a serving layer for queries over the data and a fast real time layer for low latency queries [21]. Figure 3.2 shows the overall idea of the Lambda Architecture. We will go more details on the design of each layer.

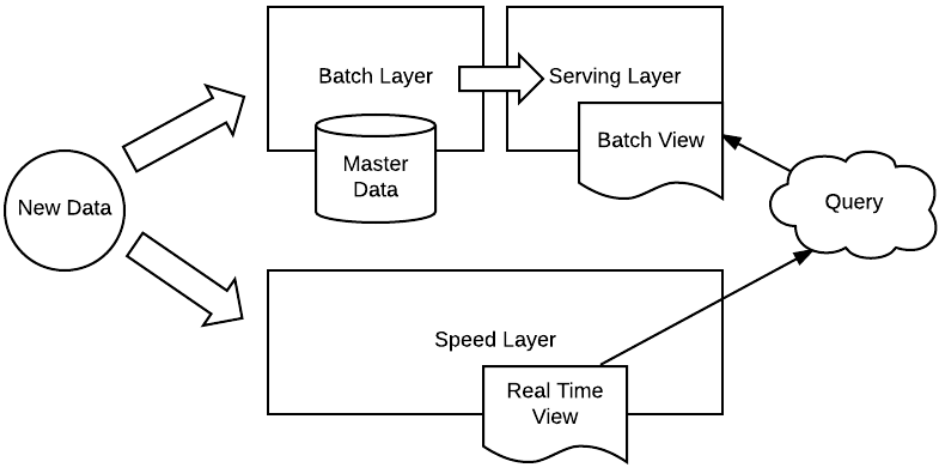

Figure 3.2: The Lambda Architecture overview

The batch layer and serving layer together do roughly what we invented earlier when discussing the requirements of our pipeline. The batch layer takes all our data and calculates a batch view of the data [21]. That means that in case we would be interested in knowing how many times a certain event occurs in the data, the batch layer would calculate this information from the data and save it to a file. This processed data would then be a new view to our data. The batch processing layer is built using specific batch processing systems. These systems allow the user to write arbitrary queries on data as the system was a single threaded program, however under the hood the program execution will be distributed [21]. The distribution is done with so called MapReduce paradigm and clusters. The distributed computing will significantly speedup the processing time compared to single threaded program.

The batch processing solution outputs the data for example to a text file. To be able to run queries over this data we must move the data to a database solution that can index and serve our data [21]. This database can be called a serving layer. This database needs to support fast random reads but no random writes. Therefore that database can be read optimized and relatively simple. The serving layer is responsible for constantly reading in the new batches from the batch layer. As we pointed out earlier the batch layer will essentially run through the whole data. Therefore the all data in the tserving layer will be replaced by every new batch.

The batch and serving layer already almost completely respond to our re-

quirements. We can process the data and run arbitrary queries over it. However we are not willing to run the batch layer constantly when the data updates, due to the latency and resource costs. Due to that the Lambda Architecture suggests a speed layer for offering the view for the most recent data [21]. Essentially the speed layer offers the same functionality as the batch layer just with the difference that the real time layer only looks at the most recent data. By most recent data we refer to the data that has not yet been processed by the batch layer. Lets say if our batch layer would run once a day, all data collected after the latest batch run will be most recent data handled by the speed layer.

The speed layer functionality also differs a little from the batch layer. Whereas the batch layer always recalculates everything from the scratch, the speed layer tries to optimize this process by updating its view on the data [21]. For example if we want to calculate certain events in the log data, the speed layer would keep a count of how many times the event occurred and each time the same event occurs again the real time layer would increment the event count. Both the application logic and database are more complicated on the speed layer than in the batch layer. However there is also less data to be handled and the results of the speed layer are only temporary [21]. Therefore possible bugs in this layer have less effect on the system.

The final step when calculating the feature from the data is to merge the result from the speed layer and the result from the batch layer. How this is handled depends from the query and other implementation details. In our example of event count this would mean that we query the count for all previous events from the batch layer and then sum it to the sum of all recent events from the speed layer. The sum will then be our total event count.

Depending on the use case, the results of the layers however do not need to be necessary merged but the layers can serve separate purposes. In addition to our case of calculating features, the Lambda Architecture can be applied for example for creating dashboards on data [16]. That means that the batch layer can offer a view on history of the data for example an average over the history of events and the speed layer can offer a view on the most recent data for example average of the events for the previous day. The speed layer can also be used for example reacting events in the data in real-time and the batch layer for providing statistical overview on the data [31].

The Lambda Architecture can be implemented in various ways, as the Lambda Architecture does not take a side on implementation details of a system. In this theses we will implement the Lambda Architecture using Amazon components. In addition to the architecture presented here, our pipeline will also need to contain the integration to Amazon Machine Learning service. Our pipeline will be introduced detailed in Chapter 4.4.

It is essential to notice that the Lupapiste service and the other circumstances will evolve over time and that will effect to the accuracy of our predictions. In a long term our pipeline should also update our machine learning model with new training data. For simplicity we will however in this Master's Theses concentrate on the real time prediction pipeline.

Chapter 4

Implementation

4.1 Data preprocessing

We will use two different data sets in our research: the Lupapiste usage logs and the operational data from the Lupapiste -database. Both of these data sets have been taken from production environment and they contain data from the past three years.

The log data contains a log entry for each action a user has taken in the user interface. For example each time a user fills a field in an application form a log entry is created. Each log entry contains the time stamp of the log entry, user name and role, the application id and action name and target. The completed list of features with descriptions is shown in Table 4.1.

The original logging data also includes the target field user entered data. However, since this sort of data might contain confidential information such as applicants names or social security numbers, we will filter out this data already in this phase. In the context of public sector storing and transferring peoples personal data is usually restricted by law[1]. By leaving out this sort of confidential data, we can freely upload our data to third party service provider such as Amazon.

The operational data in turn contains one row for each application. Each row tells the application current status, create date, submit date, possible canceled date and verdict date. The description of the operational data is shown in Table 4.2. The operational data is fetched from the Lupapiste -database in JSON-format. Since the JSON-format is unpractical for further data analyses, we will convert the data to CSV-format by using scripting languages.

Both of the data sets, log data and operational data contain the appli-

[1] http://www.tietosuoja.fi/en/index/rekisterinpitajalle.html

Column	Type	Description
datetime	datetime	The time and date of the log event
role	categorical	author or applicant
user name	text	The user name
application id	text	The unique application id
municipality	text	The municipality code
action	categorical	The action user was taking. For example uploading an attachment.
target	categorical	The target of the action. For example the type of the attachment.

Table 4.1: Lupapiste service usage logs

cation id, which makes it possible to perform a join on the data sets. That means we will create a complete new data set, containing information from both of the data sets. In the following chapters we will use this method to utilize data from both of the data sets.

4.2 Explorative data analyses

Our log data set contains lots of information about the application and the application filling process. It is likely that not all the data will be significant thinking of our target. We will next go through our data and derive the significant features.

Processing time

We start our analyses by defining our target. We aim to predict the time that has gone from the point a user submitted the application to the time the verdict was given. Both of these time stamps are already as such available in the application operational data so we will create this new feature by subtracting the submit time from the verdict time. We will call this time from now on a processing time. At this point we will need to filter out all applications that have not yet been given a verdict, since we can not calculate the processing time for them.

Column	Type	Description
applicationId	text	The unique application id
municipalityId	text	The unique municipality id
operationId	category	The application type. E.g. House
createdDate	datetime	The date application was created
submittedDate	datetime	The date application was submitted
verdictGivenDate	datetime	The date verdict was given
canceledDate	datetime	The date the application was canceled
state	category	The application status, which is one of the following: open, submitted, sent, complementNeeded, canceled, appealed, verdictGiven, constructionStarted, closed, extinct
lat	number	The latitude of the application target location
long	category	The longitude of the application target location

Table 4.2: Operational data from Lupapiste -database

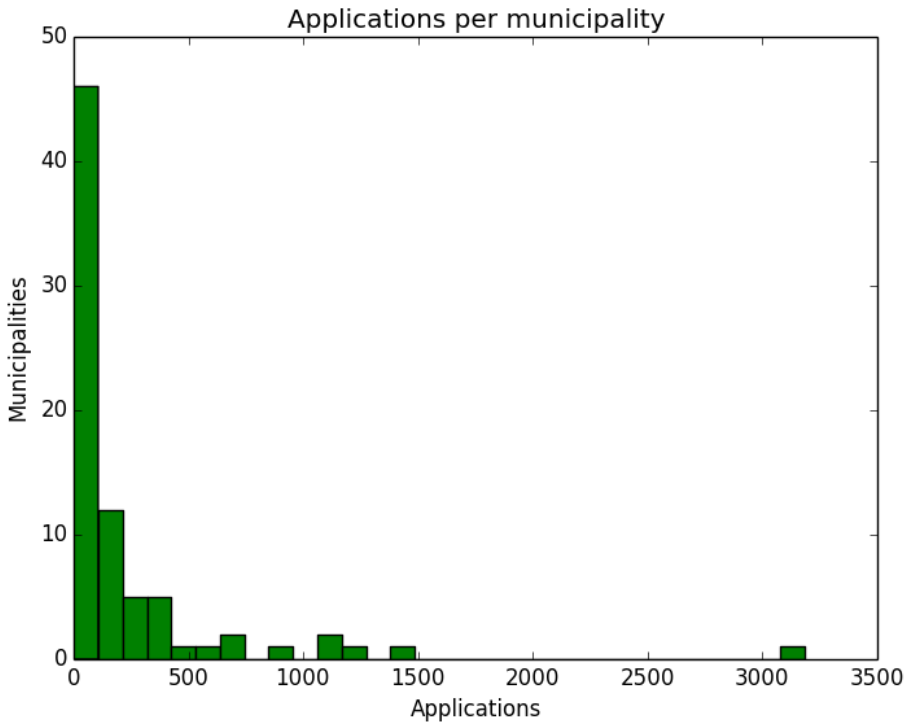

Figure 4.1: The histogram of applications per municipalities. Majority of the municipalities have less than few hundred applications altogether

Filtering the data

Our first assumption is that the municipality where the application is created would have effect on the processing time. The municipality is directly available in the operational data set for each application so we can simply add it to our feature set. When we examine the data, we however notice that there is big variation in adaption of the service over municipalities. Out of 78 registered municipalities in the service more than half have less than one hundred applications ever submitted (Fig. 4.1). Since we are worried that municipalities with less than a few hundred applications are not comparable with municipalities with thousands of applications, we filter out all but the five biggest municipalities in terms of the applications.

Next we plot the applications of the five biggest municipalities and their processing times in a box plot manner (Fig. 4.2). The plot box presents the majority ($\approx$ 60%) of the data in a box, with a line at the median and "whiskers" for the values out side this range. From the plot we can try to figure out the distribution of the processing times per municipality and the

correlation between municipality and processing time.

The plot however is not really informative, since the processing times are distributed over a big range and the majority of the values are pushed to bottom of the picture. It seems that while the majority of the applications go under $1 * 10^7$ seconds (four months), there are some applications that might even take years. We assume these being big complex construction projects operated by the public or the private sector such as a shopping center or a municipality hospital. It is not easy to see any difference between municipalities in this plot.

We decide to simplify our problem by filtering applications by application type. We assume that different types of applications go through very different processes and therefore their processing times are not really comparable. Different applications types might be effected by the same variables in different ways which would make our model relatively complicated. To avoid this, we filter applications so that we only leave house ("pientalo") -types of applications to the data. We choose this application type, since it is the most popular application type among our applications.

It is good to notice though, that by limiting our applications to the house applications and to the five biggest municipalities, we significantly cut our data set and might effect the quality of our machine learning model. Each of municipalities selected here has however 100-500 house -applications, so we consider that being enough for building a linear regression model. We filter the applications by application type and then plot the applications in the similar manner as before to Figure 4.3.

In Figure 4.3 we can finally clearly see the correlation between municipality and processing time. The municipalities with code B and code F seem to generally have clearly longer processing times than other municipalities. With municipality B, the fastest processing times go to similar numbers than the processing times of municipality E, however the average of times is clearly above of the times of municipality E.

Filling time

The next assumption we have, is that the filling time of the application would have effect on the processing time. We assume that the applicants who take longer to write their applications would write better applications than people writing their applications fast. We form the filling time by taking the start time of the application and subtracting if from the submit time of the application. We add the filling time to data which has been filtered as described in previously. We plot the processing time as the function of the filling time (Fig. 4.4).

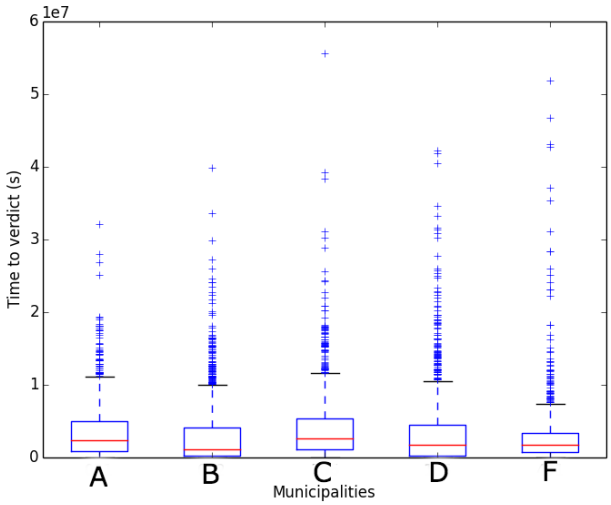

Figure 4.2: The processing times per municipality

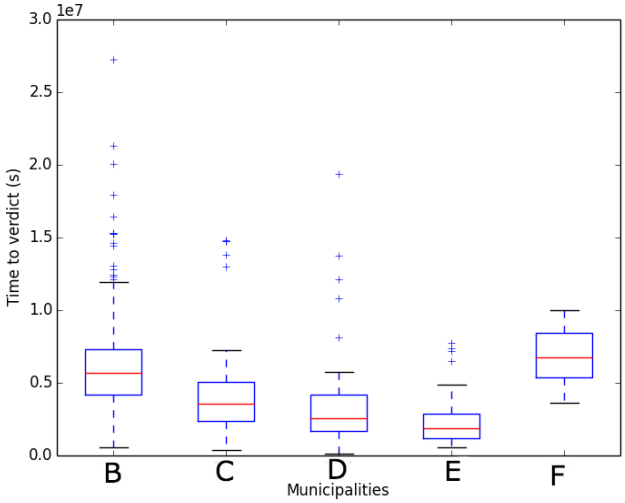

Figure 4.3: The processing times (time to verdict) per municipality with house -type of applications

If there is any correlation between the filling time and verdict time, it is not visible from Figure 4.4. What makes examining the data difficult, is that most of the data is packed to lower left corner of the picture, since few extreme values force axes to widths that are too wide for most of the data. These extreme values not only make the plot difficult to read, but also might pose a challenge to our linear regression model. To solve this problem, we take a base hundred logarithm from the processing time and the filling time and plot the values again (Fig. 4.5).

With the logarithmic function, we are able to get the values to spread over wider area, which makes the figure more readable. However still no strong correlation between the filling time and the processing time is visible. The only thing visible in the picture is the correlation between the municipality time to the processing time, which was already spotted earlier. The feature does not seem to have correlation with the target based on this research, nevertheless we will test it in a later phase with the Amazon Machine Learning service.

Application month

Our next assumption is that the month of the creation date of the application would have correlation to the processing time. We have the assumption that some months are slower, since during holiday season there will be less people working in municipalities. We add the creation month of the application to the data and plot the data in the box plot manner to Figure 4.6. From the Figure we can see that the longest processing times are at summer and at the beginning of the year, however the correlation does not seem to be strong. It seems that the house applications being complex construction processes they depend on many external variables and do not depend strongly on time of the year.

Running month and applications per month

Next we come up with two additional assumptions from the data. We assume that the running month starting from the beginning of the Lupapiste service lifetime would have an effect to the processing time. We assume that over the time municipalities become more fluent in handling the applications, which makes the processing times faster. Our other assumptions is that amount of applications per month might effect the processing times. We assume that in the month, where there is lots of applications the processing times might be slower than in the months where there is less applications.

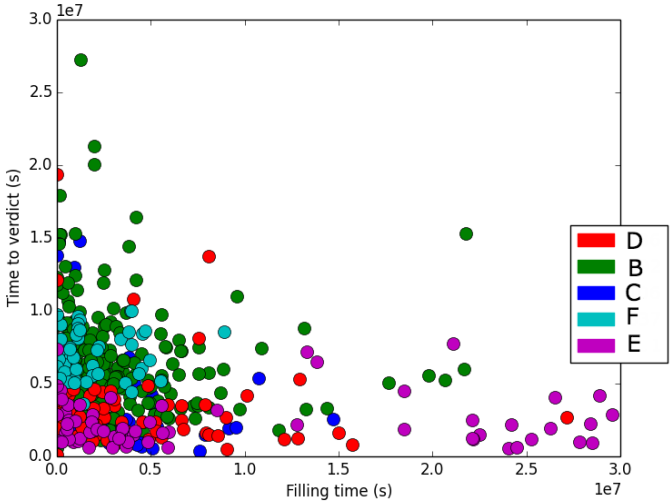

Figure 4.4: The processing times (time to verdict) as seconds per filling times as seconds. Municipalities separated with color.

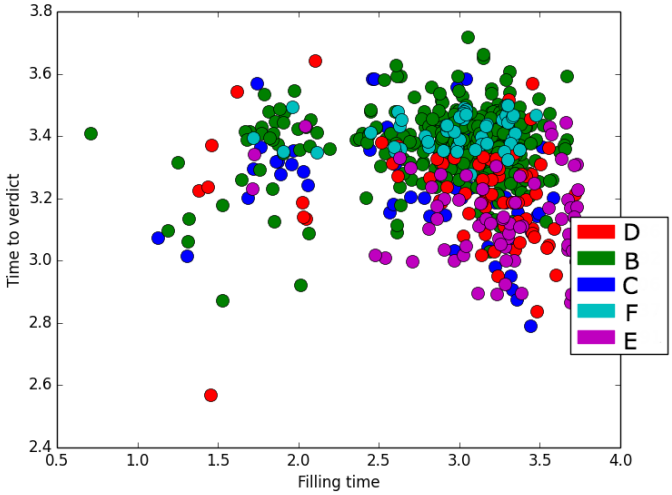

Figure 4.5: The base hundred logarithm of processing times (time to verdict) per filling times. Municipalities separated with color.

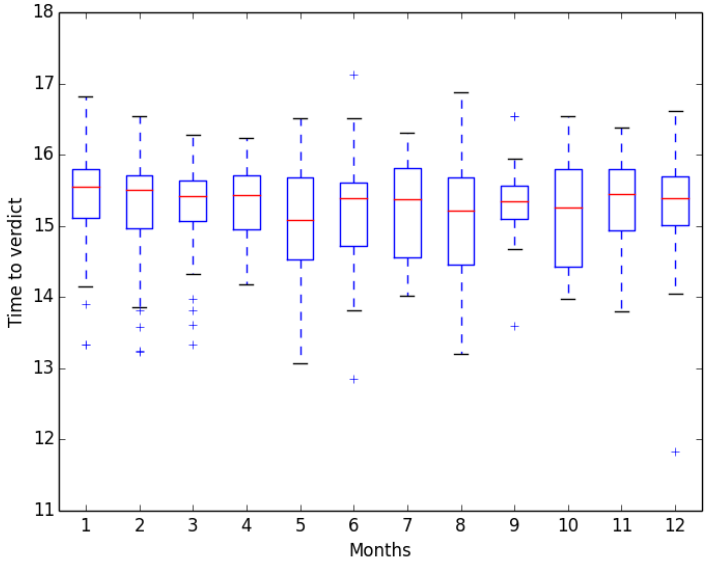

Figure 4.6: The processing times (logarithmic values) distribution over start months presented as box plot

We create Figure 4.7 to examine our assumptions. From the figure there is no clear trend to be seen in the development of processing times over time. However with some municipalities we can see a correlation with the amount of applications per month and the processing times. Especially with municipality E there is clearly a peak both in applications per month and processing times between the months 30 (06/2015) and 35 (11/2015). It is however difficult to say whether one feature caused a peak in the other or are both peaks caused by same external variable. We will however take the applications per month as a feature to our data set to test it with Amazon in the next phase.

Action count

So far our assumptions have mostly derived from the operational data. We will next take a closer look at the log data. Our assumption is that the bigger amount of log events would result in faster processing times. We assume that the more people edited their applications the better the application quality would be. We plot the processing time per application log action count (Fig. 4.8). We filter out all the log events that happened after the submit.

In Figure 4.8 we see correlation between the log event count and the processing time but different from what we expected. The more log events there

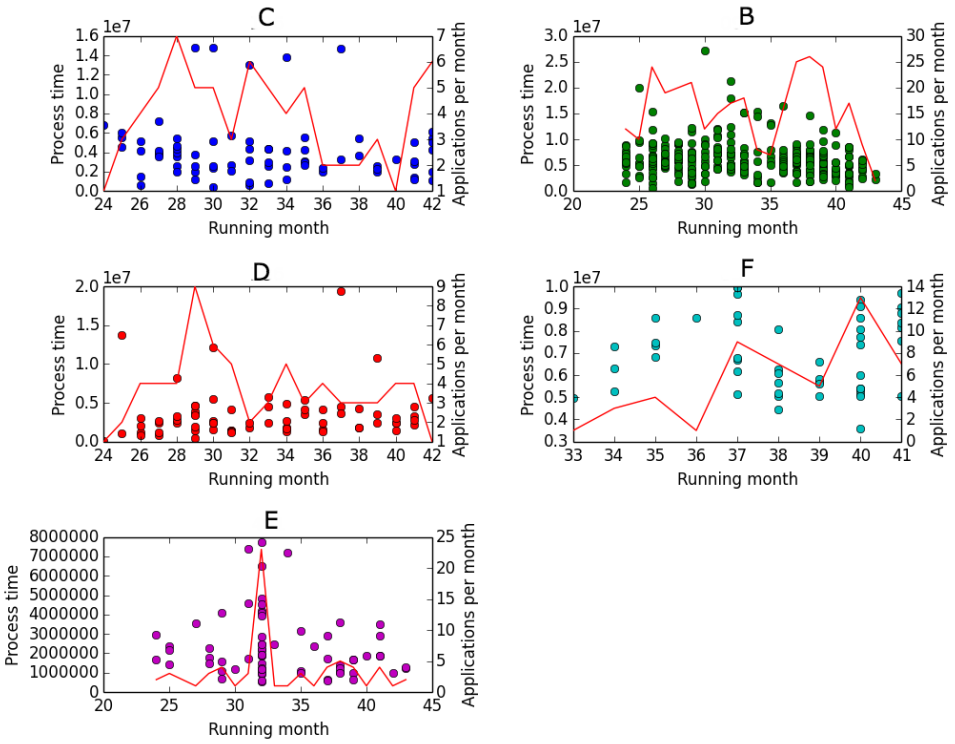

Figure 4.7: The processing times per running month (the month starting from Lupapiste -release month) and the amount of applications per month.

are the longer are the processing times. For a reference we also try out different types of log events. We plot the processing time as a function of the amount of events related to uploading attachments and the processing time as a function of actions that are related to updating the competence information of the lead designer of the process (Fig. 4.9). In both of these we see similar correlation between the action count and processing time as with the log event count. It seems that the more complicated the application is the more log events is spans and the longer it takes to process.

4.3 Machine learning model

After we have come up with promising features from the data, we will build our linear regression model with Amazon Machine Learning service. We will first upload our data set to Amazon S3 bucket. Then we create a machine learning model from the data using the Amazon Machine Learning service.

The service user interface guides us through the process. The service asks us to define the types for our features (numerical, categorical, binary or text). We will mark the municipality and month as categorical features and others as numerical features. The service also asks us to define the target feature. For all our experiments in this Chapter, we will use the processing time as the target feature.

The Amazon Machine Learning service does not give much freedom for customizing the machine learning methods. The only available algorithm is the linear regression algorithm. For advanced users there is the possibility to customize some of the algorithm parameters: number of passes over data and regularization type and amount. Tuning the regularization type might become useful with data sets with significant amount of features, so it is not likely to be useful with our data. The number of passes might be something we would benefit about, since it might be useful especially with smaller data sets. This one however will increase the time and costs of building the model and it is not likely to make big difference with our data. So with following cases we will go with the default parameters.

Once we are happy with all the settings, we will ask Amazon to build a model based on the data. With our data volumes of hundreds of rows of data the service takes about two-three minutes to create the model. After creating the model, we will examine its performance. The service will evaluate the model performance using the evaluation set we define. We will randomly split our data to training and evaluation data so that it uses 70% of the data for training the model and 30% data for evaluating the model.

For evaluating the model the service will calculate predictions for the eval-

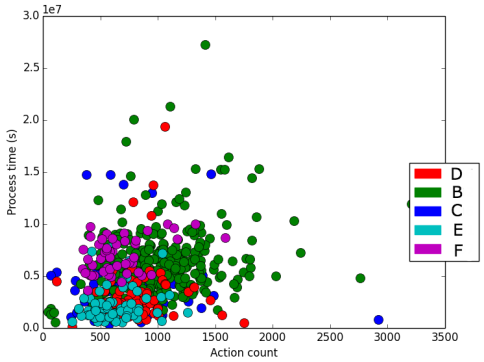

Figure 4.8: The processing time per action count

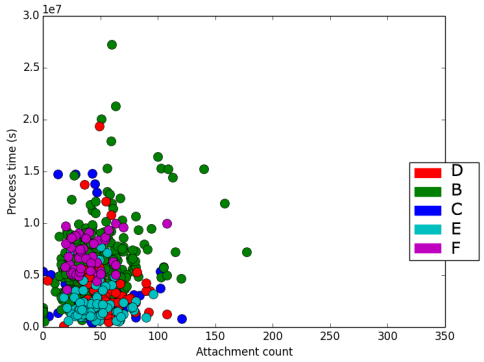

Figure 4.9: The processing times per attachment count

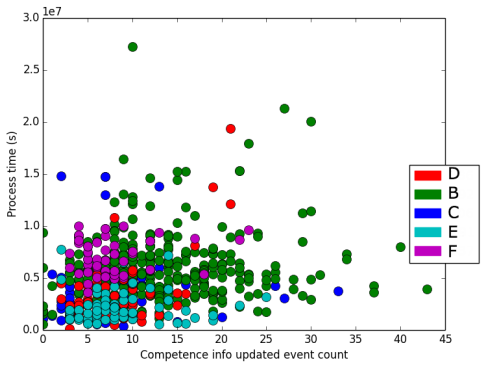

Figure 4.10: The processing times per competence info updates

uation data set using the model built with the training data set and then calculate the RMSE for the predictions. The service will then in a similar manner calculate a baseline RMSE using a baseline prediction model, which simply uses the average of all values as prediction. By comparing our RMSE to the baseline RMSE we can get some idea how well does our model perform. However, we should not rely too much on this comparison, since even though our solution would be better than the baseline solution, it might still not be enough good for any practical uses cases.

We will build the model feature by feature, so that we can verify our features and assumptions from the data simultaneously. We start with a feature set that has only the processing time and municipality for each applications. The Machine Learning service builds the linear regression model and gives an RMSE-value 3 552 120 seconds (41 days). To understand this error we need to examine our data little more. From Figure 4.3 we get a idea of the distribution of the verdict times. The median values are around $0.5 * 10^7 = 5\ 000\ 000$ (57 days). That means, that RMSE of our model is about 70% of the median value meaning that for most of our data the error would be 70% of the value. That means that when the real processing time is 57 days the prediction might be anything between 16 - 98 days. It is easy to understand that this prediction is not accurate enough for any practical use. The prediction is even slightly worse than the simple baseline solution that would be 3 544 602 seconds.

We continue to develop our model by adding the next feature, filling time to the model. By adding this feature, we are able to improve our model slightly and we get a RMSE 3 502 038s (40 days). The error goes down by 10 hours compared to previous error. For comparison we also build a model that only has the filling time and the target. With this model the results are worse than the baseline. Seems that the filling time only has effect on processing time, when the information is combined with municipality. In Figure 4.11, we see the processing times per filling times per each municipality. As we see, for some municipalities the processing times seem to have more correlation with filling times (B, F), whereas for other municipalities the values are more scattered. Therefore it seems natural that these features must be combined to bring up reasonable results.

Next thing we do is adding the start month to the feature set. The RMSE is slightly greater than with previous features combination as shown in Table 4.3. It questionable if start month brings any value to our model. We will however keep the start month with us in the following experiments to find out whether it would improve our model in some other feature combinations.

Next we add the applications per month feature to the feature set. By including this feature we are able to get our RMSE down by about 10 hours

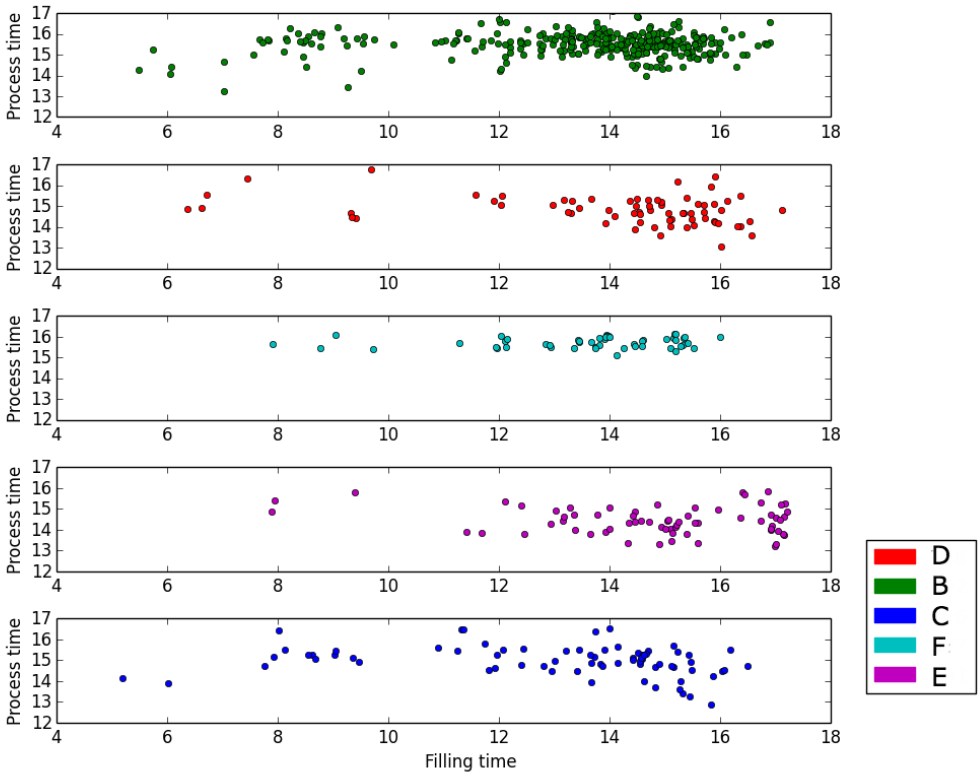

Figure 4.11: The processing times (logarithmic values) per filling time per municipality

Features	RMSE (s)	Difference (s)
baseline	3 544 602	0
municipality	3 552 120	-7
filling time	3 821 804	- 277
municipality, filling time	3 502 038	42 563
municipality, filling time, start month	3 506 041	38 560
municipality, filling time, per month	3 480 556	64 045
municipality, filling time, start month, per month	3 461 313	83 288
municipality, filling time, start month, per month, action count	**3 458 713**	**85 888**
municipality, filling time, action count	3 502 739	41 862

Table 4.3: RMSE:s with different feature sets

compared to feature set only containing municipality and filling time. It seems that the applications per month has correlation with our target. When we however add the start month to the feature set we are able to get even smaller RMSE than with only per month -feature (Table 4.3). It seems that the start month feature alone can not improve our model but when combined with the applications per month feature it improves our model.

Finally we add the log event count as one of the features in the model. We run it together with all other features introduced earlier in this Chapter. We are able to get small improvement to our model. We try out a different combination of features and drop out the start month and the per month. Without those features the RMSE grows significantly (Table 4.3). It seems that the action count might bring some improvement to our model, but only combined with other information such as start month or applications per month.

Our final results are presented in Table 4.3. In the table the feature set and the RMSE and the difference to the baseline RMSE are presented. When examining the results we notice that even the best results are only one day better than the baseline. After all our model is still not much use in real production environment. It is however good to keep in mind that our scenario is relatively challenging and there is no quarantine that even the best possible solution out of this data would be significantly better than our cur-

rent solution. We will continue our implementation and discuss about further possibilities to improve the model in Chapter 5.

In Chapter 4.4 we will implement the pipeline for predictions. For show case we will choose features municipality, the filling time and the action count for our model. By using these three features we can show how feature data is collected and discuss about the structure of our pipeline.

4.4 Pipeline implementation

Once we have the model in the Amazon Machine Learning service we want to integrate it to the Lupapiste service to show the real time predictions in the user interface. The goal is to show to the user the prediction of the application process real time after user has submitted his application.

As stated in Chapter 3.3 we suggest the Lambda Architecture for handling the real time predictions. We will here discuss the implementation of the speed, the batch and the serving layers. We will leave out the functionality of updating the machine learning model since it is not necessary for a minimum implementation of the pipeline. If wanted, the model can be updated manually following the same process as described in Chapter 4.3. However given that the current model has the data from past three years, model update needs to be considered earliest after several months.

We will set up the components on the Amazon cloud environment and test the latency of our predictions and the overall suitability of this pipeline to our problem. We will setup the pipeline in the separate test environment. That means we will not get an experience from running the pipeline in real production environment and can not therefore confirm the scalabily of our pipeline in production use. We will however learn about setting up the pipeline and discuss about the implementation of it.

To understand the design of our pipeline we will go through our requirements detailed. Once user submits the application we want to offer a prediction for the user based on his application. Amazon Machine Learning service offers a real time prediction endpoint for requesting the prediction. However as explained in the previous chapters about selecting the features and building the model, some of our features are not directly available in our data but they must be derived from the data to the features. We want to use the count of log events for the application as one of the features. To achieve that we should go through the log files on a server each times we want to show a prediction to the user. The log files are together several gigabytes and alone reading the files to memory will cause a latency, not to talk about reading through files and fetching the information we want.

To avoid latency and possible errors while grepping and reading the log files we will need to go for more intelligent solution. We will index our log data to Amazon Relational Database (RDS). Any relational database from any provider would work here, but we will use only Amazon components when building this pipeline to offer smooth communication between components and to take the advance of the cloud environment. In addition to relational database Amazon offers also several non-relational database solutions, which are specifically designed for fast data access. However the relational database is suitable choice for us, since the relational databases are designed to aggregate operations such as *count*, which is our use case here.

The relational database is the speed layer of the Lambda architecture. Our speed layer design differs slightly from the suggested design in Chapter 3.3. In the Lambda Architecture the speed layer supposedly contains application logic constantly updating the view on the data. In our solution we constantly query the data from our database. This is trade off between complexity and scalability. Our solution is not the highly scalable but it is simple and therefore less prone for errors and it works as long as the database size does not grow too much.

To get the log data to our relational database we will need to modify the Lupapiste service code. We will edit the code so that it pushes each log event to our relational database immediately after the log event is created. The advantage of method is that we have full control over when the log event is indexed to our database. The downside is however that we need to modify and maintain the code of the Lupapiste service. We will discuss about alternative implementation solutions in Chapter 5. In Chapter 4.1 we described the two data sets: log data and operational data. When saving the event to our relational database we can do the merge already while pushing the log event to the database. That means that we attach the operational data to each log row while pushing the event the to database. In similar way than described in Chapter 4.1 we will for information security reasons need to filter the data by omitting the data user inputted to the field.

When pushing a log entry to the real time layer database we will also push the log entry to the master database (Fig. 4.12). The master database is part of the batch layer which calculates the view to all of the data and updates the serving layer. Once the batch process has finished, we can remove the data from real time layer database. That way the queries to the real time database stay relative fast, as the database stays small. The master database will stay unmodified and no data will ever be removed from there.

The schema of the log data table both in the real time and master databases is presented in Listing 4.1. As we see the schema contains both the features from the operational data and from the log data as described earlier. We also

see that there are several fields in the schema that are not part of our machine learning model for instance *longitude*, *latitude* or *canceled date*. It would not be necessary to save them to the database. We will however also save them both to the real time database and master database in case we later want improve or develop our machine learning model further and would also be need any of these fields as features. By saving all the data already now there will not be later a need for further modifications of Lupapiste service code or this schema.

Listing 4.1: The schema for log data -table

```
CREATE TABLE log_data (
    datetime timestamp,
    applicationId    varchar(40),
    user varchar(40),
    role varchar(40),
    action varchar(40),
    target varchar(200),
    municipalityId varchar(40),
    state varchar(40),
    operationId varchar(100),
    submittedDate timestamp,
    sentDate timestamp,
    verdictGivenDate timestamp,
    canceledDate timestamp,
    isCanceled boolean,
    lon varchar(40),
    lat varchar(40)
);
```

Overall the process of offering the real time prediction to user is described in Figure 4.12. In the right upper corner we have the Lupapiste service. The arrow from Lupapiste to real time database describes the log data flow from the Lupapiste to the real time database and to the master database. The merge symbol describes the phase where log data is enriched with the operational data before pushing it to the databases.

Further in Figure 4.12 the arrow from the Lupapiste component to the Lambda component is the request for the real-time prediction, triggered by the user submitting the application. The Lambda component here is an instance of Amazon Lambda service. Amazon Lambda service is a service provided by Amazon and it should not be confused with the Lambda architecture described in Chapter 3.3. Lambda service is stateless computing platform, which runs Python, Java or JavaScript -code that programmer offers. The

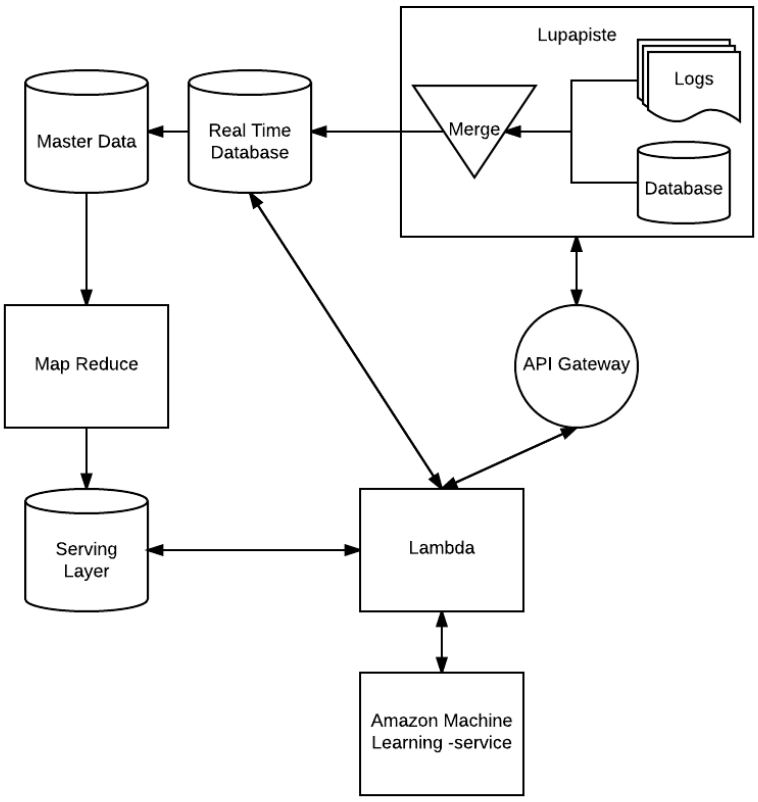

Figure 4.12: The data pipeline system architecture diagram

code is configured to run on events such as a new data entry in a database or as in our case configured to run on API-calls via Amazon API Gateway. After running the code the Lambda will clean the running environment and shut itself down until the next event. The programmer of the service does not need to set up the server or container for the code to run, but Lambda will handle all of these automatically.

The call to Lambda goes through Amazon API Gateway component that triggers the Amazon Lambda component. The API Gateway wakes up the Lambda component and delivers the request from the Lupapiste service to the Lambda. The API Gateway will also return the response from the Lambda service to the Lupapiste service acting as a proxy to the Lambda.

The Lambda contains the business logic of our pipeline. The whole Lambda -code can be viewed in Appendix A. After being invoked the Lambda will read the request body transmitted from the Lupapiste service. The body contains the application id, start date, submit date and municipality of the application in JSON-format as shown in Listing 4.2. All of this data is stored as such in the Lupapiste service database. That means we can easily query this data from the database when making the request for the prediction in the Lupapiste service.

Listing 4.2: Request Body from Lupapiste to API Gateway

```
{
"applicationId": "LP-710-2016-00288",
"municipality": "018",
"start_time": "2016-01-25 11:50:09.570000",
"submit_time": "2016-02-18 13:31:38.434000"
}
```

The Lambda component will make a query to the real time database to read how many log items there are for given application id. Latest here we see why it is crucial to update the real time database immediately after the log event. If our log events are not updated to the database this query will bring smaller result than the actual count of the log events and therefore give biased prediction to our application processing time. After that Lambda queries the serving layer to get the history view on the data. Finally Lambda combines these results by summing the log event counts from the real time database and serving layer for a total event count.

The other features in our chosen model are the filling time and the municipality. The municipality of the application is provided in the request so no further actions are needed to get this value. The filling time can be calculated be subtracting the start time from the submit time provided in the request. After collecting the features we will fetch the real time prediction from the

Amazon Machine Learning service. After that we return the prediction to the Lupapiste, which shows the prediction in the user interface.

The arrow from the master database to the MapReduce describes our batch layer functionality. The MapReduce component is a Hadoop instance that runs in Amazon cloud. It takes in our data and aggregates the data in a similar manner than in the speed layer. It reads in all enriched log data from the master database and calculates the application count for each application by grouping the log events by the application id. For configuring the Hadoop to execute the described functionality we use Apache Pig script language. Pig script language will let us write our logic using SQL like queries, which will be run distributed in the Hadoop instance [27]. Our Pig script for configuring the Hadoop instance can be seen in Appendix B.

After finishing the batch process the Hadoop instance writes the results to text files in Amazon S3 bucket. To get data to our serving layer, we need to import data from text files to our serving layer which is essentially a key-value database in Amazon cloud. In this process we use Amazon pipeline components that are specifically designed for converting data between text files and database. For our serving layer we use Amazon Dynamo database. Amazon Dynamo database offers a simple key-value interface, where objects can be fetched by their corresponding key. By design the Dynamo database emphases availability over the data consistency, making it highly available data storage [10]. This provides us fast and easy access to our data in the serving layer.

The part missing from the architecture diagram is the logic which triggers the batch process and empties the real time layer. These are however implementation specific questions and not performance critical in a similar manner than the speed layer. Therefore we will not go to the implementation of those components. Essentially these can be handled with Amazon pipeline components.

Overall our pipeline performance is satisfying. We are able to get our predictions in few seconds even when the real time database, which is the performance wise potentially the biggest bottle neck, has 50 000 rows. Overall setting up and configuring all the components was relatively easy and overall the architecture is easy to maintain, as very little custom or ad hoc components are needed. We will discuss further about our data pipeline in Chapter 5.

Chapter 5

Discussion

Overall our data pipeline performed well and was relatively easy to set up. The Amazon Machine Learning service turned out to be a good solution concerning to our pipeline. Instead of building a complicated service with machine learning libraries and several data processing solutions, we got a ready build machine learning service. That made building the real time predictions pipeline significantly easier. Also the Lambda Architecture supported well our goal of building a real time pipeline. The pipeline separated the fast real time layer from the slow, big data batches, offering a fast but robust setup for real time predictions.

Our pipeline offers high scalability for requesting predictions. The Lambda component handling the requests scales almost limitless. As no state information is required for the Lambda component, several Lambda components can easily run simultaneously. The Amazon Machine Learning service promises to deliver up to 200 predictions per second, which makes it possible for us to serve up 200 people submitting their applications simultaneously. Our serving layer, Amazon Dynamo database also offers fast access on the history view of the data. According to Amazon, even up to 20 000 simultaneously read operations are allowed.

However the amount of connections to the relational database components of our pipeline is limited. Depending from the price category, the maximum amount of connections is limited to hundreds or to thousands of concurrent connections. Unlike other components in the pipeline the relational databases need to handle random write operations, as the log data is written to the database, as user fills in his application. That makes the relational databases a potential bottle neck of our pipeline. If the amount of connections exceeds the maximum amount allowed by Amazon, it would lead to possible delays in the process or even in data loss.

To avoid this bottle neck in our pipeline a service specialized for handling

fast data streams such as Apache Kafka could be used to read in the log events [1]. Essentially the Kafka system reads in stream of events and stores them in internal queue which applications subscribe to consume the events [18]. An application subscribing Kafka event queue could then read events from the queue and process and write the data for example on key-value pair database to offer the real time view on the data. An other application subscribing the events could then handle writing data to the master database. This way we could avoid possible bottlenecks caused by the relational databases. However as we did not test our pipeline in real production environment, we did not have possibility to experiment with this matter. We will therefore leave scaling the log event writing for further research.

Other matter to discuss about in our pipeline design is, that it requires modifications to the existing Lupapiste -code. The modifications to the service code need be maintained over the service lifetime and if requirements for the pipeline evolve the code needs to be updated. This requires extra work and might lead to errors and bugs in the system. To avoid modifying the code we could have used a specific log aggregation tool to read and index the log files from the server. Actually the Lupapiste service already has a Splunk log aggregation tool configured and running. We could have run queries over the data using the Splunk tool instead of using a relational database for indexing the log data. The flaw of this approach is however that we do not have direct control on when the Splunk indexes the log data. If the data is not indexed immediately after the log event, that might cause possible bias in our predictions. Therefore our solution turned out to be fluent for our case.

Building the machine learning model with the Amazon Machine Learning service also had its challenges. Despite of our efforts and logical reasoning even our best feature combination gave predictions which were only one day better than the baseline. There are however many factors that can explain our results. According to Lupapiste service specialists from Solita, the biggest factor effecting the application processing time is the personnel. For example the overall work situation of personnel handling the applications has great impact on the processing times. The work situation of the personnel is however not directly available in our data. We tried to tackle this by using the applications per month as one of the features in our model, however with no significant success. It seems that the personnel handling the applications are also involved in several other tasks which are non-related to Lupapiste service and therefore not available in our data.

Other factor related to the personnel is people being away from work. Naturally this is difficult to predict since people might be away from work due sick leave or due other non-predictable causes. We tried to take this factor into account by selecting the month of the application to one of the features.

We assumed that some months might be slower due holiday seasons. We were able to get some improvement to the model with this feature, but again no significant change. In addition to holidays and overall workload there are also many other things about personnel that might effect the application processing time which are just not available in our data. These could be for example changes in the personnel, the personnel competence and habits.

One more circumstance noted by the Lupapiste specialist as one of the reason to delayed processing times is the quality of the attachments. The attachments might not simply be at the level what is expected and do not show the information what is needed. The quality of the attachments is however obviously not visible in the log data, so can not be used as feature. Other thing pointed by the specialists was that the quality of the answers in any point of the application is generally poor and that effects the processing time. Due information security reasons as stated before this user inputted content had to be filtered out for our experiments. Given that Amazon is trusted cloud provider by many public sector instances in US it might not be impossible to negotiate possibility to upload the user inputted content to Amazon cloud environment[1]. Especially it might have been possible to upload the data to cloud environment, if we would have kept the data in the area of European Union. That is possible due Amazon offering several regions for its services, some of them being located in the area of European Union. It is however good to notice that even though we would have included the user inputted text to our data, analyzing free text is not trivial and would have required additional methods and research.

Adding comprehensive logging data or utilizing various data sets for example data from the human resources systems of the municipalities might have improved our model. Also adding some information about the attachments might have given us more insight to the predictions. However it is good to notice that in any case our scenario is challenging. By only utilizing the information that is available the moment user submits the application we limit ourselves a lot. For example if we would wait until the moment one of the personnel assigns the application for himself we could use that information as part of the prediction and possible improve our predictions. However then we would needed to give up the real time aspect from our predictions.

Other factor we can discuss in our predictions quality is the Amazon Machine Learning service. Many ways Amazon Machine Learning service turned out to be suitable for our problem. The usage and adaption of the service turned out to be really straightforward and the performance of the real time predictions turned out to be fast. However in the machine learning point

[1]https://aws.amazon.com/compliance/resources/

of view the service was limited to one algorithm, linear regression. If the service would have offered other machine learning algorithms it would have been possible to explore with different methods and possible get better predictions quality. Also for experimenting with the data the service turned out be relatively ungraceful. The service utilized surprisingly little automation in this matter. For example when wanting to try different sorts of combination of features user had to upload all different feature combinations manually. By adding a relatively simple feature that automatically trains and evaluates different feature combinations from feature set would have dramatically improved the service usability. Also chance to train a linear regression model locally using tools such as Matlab or R and to upload that model to Amazon Machine Learning service would have been an interesting possibility.

The prices of the Machine learning service were relatively decent for our needs. As doing this research in December 2016, machine learning model building costed 0.42$/hour. The real time predictions costed 0.0001$ per one prediction. With our volumes of data that would mean few dollars per day. Other costs for the pipeline come from the Lambda component, the MapReduce component, the API Gateway and the databases, from which the databases are the most expensive components in the system prices moving between 10-500$ per month depending the size of the database and other design details. With our data volumes our pipeline solution would cost around 100-400$ per month.

As for further research we have few proposals for improving the predictions quality. We would suggest combining more data sets for the predictive model for example data sets from other municipality systems. The other data sets might be able to give insight on the missing factors such as personnel work situation. They might also provide complete new inside about construction related circumstances in the municipality area. In addition to different data sets, also different research methods could be applied. It might be useful to interview municipality personnel about the matters they think effect the most the application process. It might also be interesting to observe the municipality personnel at work through typical working week. That would give a good insight on where does the processing time of the application actually go.

For further research we also suggests researching some other promising prediction targets. For example it might be possible to create a machine learning model that would predict whether the application will be rejected or accepted after the process and tell, which are the factors effecting this. This model could then be used to warn user about whether his application will be accepted or not and give user the information needed to correct the application to the state that it will be accepted. This would however require

enriching the data with the information about whether the application was accepted or not, as this information is not available in our data set.

For further research regarding the pipeline, we suggest testing the pipeline in production or in simulated production environment. Essential would be to test how does the pipeline scale under heavy write operations and how fast do the predictions serve when the databases have extensive amount of data. Based on these tests, design improvements could be suggested for the pipeline such duplicating certain components in the pipeline or replacing some bottle neck components with complete new design decisions.

Chapter 6

Conclusions

The objective of this thesis was to bring real time predictions to the Lupapiste Web service. The aim was to show user her/his application processing time meaning the time from the point she/he submits the application to the point when municipality gives a verdict about the application. In contrast to most of the previous research, we applied supervised learning method to the Web context. The goal was to learn about Amazon Machine Learning service and about the challenges of building a real time prediction pipeline and discuss about the right architecture choice for the pipeline.

We built a linear regression model based on Lupapiste log and operational data using Amazon Machine Learning service. For information security reasons, we filtered out all the user inputted content from the log data. We chose the features to the model based on explorative data analyses. We learned that the filling time, the municipality, the application month and the applications per month have effect on application processing time. We also learned that the action count in the log files probably has some correlation with the processing time. We were able to build a model which performed better than a baseline solution, still however relatively poorly. Our model was one day better than the baseline solution. We discussed that the cause for this might have been that not all factors that effect the processing time are available in our data. These sort of factors would have been information about the personnel, their overall work situation and holidays and information about the application attachments. We also pointed out that filtering the data so that all user inputted content was left out also probably effected our prediction accuracy.

For improving the predictions we suggest further research where the Lupapiste dataset could be combined with some other datasets from other municipality systems. We also suggest experimenting with different research methods for example interviewing the municipality personnel to get clearer picture

of what is the internal process of processing applications in the municipality.

We considered Amazon Machine Learning service a good choice concerning our pipeline. The service was easy to start using and to integrate to our pipeline. Using a ready built machine learning service simplified our pipeline design. The service also had sufficient performance considering the real time predictions. However from machine learning point of view the service turned out to be relative simple. The service had only one available algorithm for building the predictive machine learning model: linear regression. It would have been interesting to explore with different machine learning methods. Also even though advertised for exploring the data, the service user interface did not really serve that purpose.

We built the real time analytics pipeline using Amazon components. We built the pipeline by the guidelines of the Lambda Architecture. The architecture worked well to our problem and building the pipeline was relatively easy as many of the Amazon components supported our needs directly and were easy to set up. We did not, however, test our solution in production so no real performance data from production is available nor production ready solution. This was left for further research. We considered the price of all needed pipeline components decent.

We learned that the pipeline design depends heavily on the nature of the data, for example, which features need to be collected for predictions. To summarize we could say that the biggest lesson from this Thesis is that it is essential to study the data before starting to build a pipeline around it. Lots of efforts and money might be lost in pipeline design and implementation if at the end no real machine learning model can be built from the data or the design does not meet the actual requirements. Even if some decent predictions would be built without any analytics of the data, it is unlikely that any company would want to invest this sort of service or show any predictions to the users in the long run, if there is really no understanding about what the predictions are based on.

Bibliography

[1] *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Warsaw, Poland, August 11-14, 2014 - Volume III* (2014), IEEE Computer Society.

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Commun. ACM 53*, 4 (2010), 50–58.

[3] Banerjee, P., Shenoy, U. N., Choudhary, A. N., Hauck, S., Bachmann, C., Haldar, M., Joisha, P. G., Jones, A. K., Kanhere, A., Nayak, A., Periyacheri, S., Walkden, M., and Zaretsky, D. A MATLAB compiler for distributed, heterogeneous, reconfigurable computing systems. In *8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2000), 17-19 April 2000, Napa Valley, CA, Proceedings* (2000), IEEE Computer Society, pp. 39–48.

[4] Borges, J., and Levene, M. Data mining of user navigation patterns. In *Web Usage Analysis and User Profiling, International WEBKDD'99 Workshop, San Diego, California, USA, August 15, 1999, Revised Papers* (1999), B. M. Masand and M. Spiliopoulou, Eds., vol. 1836 of *Lecture Notes in Computer Science*, Springer, pp. 92–111.

[5] Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., Ranganathan, K., Molkov, D., Menon, A., Rash, S., Schmidt, R., and Aiyer, A. S. Apache Hadoop goes realtime at Facebook. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011* (2011), T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, Eds., ACM, pp. 1071–1080.

[6] Cattell, R. Scalable SQL and NoSQL data stores. *SIGMOD Record 39*, 4 (2010), 12–27.

[7] Daruru, S., Marin, N. M., Walker, M., and Ghosh, J. Pervasive parallelism in data mining: Dataflow solution to co-clustering large and sparse netflix data. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009* (2009), J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. J. Zaki, Eds., ACM, pp. 1115–1124.

[8] Davidson, J., Liebald, B., Liu, J., Nandy, P., Vleet, T. V., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., and Sampath, D. The YouTube video recommendation system. In *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010* (2010), X. Amatriain, M. Torrens, P. Resnick, and M. Zanker, Eds., ACM, pp. 293–296.

[9] Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004* (2004), pp. 137–150.

[10] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007* (2007), T. C. Bressoud and M. F. Kaashoek, Eds., ACM, pp. 205–220.

[11] Eirinaki, M., and Vazirgiannis, M. Web mining for Web personalization. *ACM Trans. Internet Techn. 3*, 1 (2003), 1–27.

[12] Fan, W., and Bifet, A. Mining Big Data: Current status, and forecast to the future. *SIGKDD Explorations 14*, 2 (2012), 1–5.

[13] Ghoting, A., Krishnamurthy, R., Pednault, E. P. D., Reinwald, B., Sindhwani, V., Tatikonda, S., Tian, Y., and Vaithyanathan, S. SystemML: Declarative machine learning on MapReduce. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany* (2011), pp. 231–242.

[14] Guy, I., Zwerdling, N., Ronen, I., Carmel, D., and Uziel, E. Social media recommendation based on people and tags. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010* (2010), F. Crestani, S. Marchand-Maillet, H. Chen, E. N. Efthimiadis, and J. Savoy, Eds., ACM, pp. 194–201.

[15] James, G., Witten, D., Hastie, T., and Tibshirani, R. *An introduction to statistical learning*, vol. 6. Springer, 2013.

[16] Kiran, M., Murphy, P., Monga, I., Dugan, J., and Baveja, S. S. Lambda Architecture for cost-effective batch and speed Big data processing. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015* (2015), IEEE, pp. 2785–2792.

[17] Koren, Y., Bell, R. M., and Volinsky, C. Matrix factorization techniques for recommender systems. *IEEE Computer 42*, 8 (2009), 30–37.

[18] Kreps, J., Narkhede, N., Rao, J., et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB* (2011), pp. 1–7.

[19] Lin, J. J., and Kolcz, A. Large-scale machine learning at Twitter. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012* (2012), K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, Eds., ACM, pp. 793–804.

[20] Linden, G., Smith, B., and York, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing 7*, 1 (2003), 76–80.

[21] Marz, N., and Warren, J. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Manning Publications Co., Greenwich, CT, USA, 2015.

[22] Mobasher, B., Cooley, R., and Srivastava, J. Creating adaptive Web sites through usage-based clustering of URLs. In *Knowledge and Data Engineering Exchange, 1999. (KDEX '99) Proceedings. 1999 Workshop on* (1999), pp. 19–25.

[23] Mobasher, B., Dai, H., Luo, T., and Nakagawa, M. Effective personalization based on association rule discovery from Web usage data. In *3rd International Workshop on Web Information and Data Management (WIDM 2001), Friday, 9 November 2001, In Conjunction with ACM CIKM 2001, Doubletree Hotel Atlanta-Buckhead, Atlanta, Georgia, USA. ACM, 2001* (2001), R. H. L. Chiang and E. Lim, Eds., ACM, pp. 9–15.

[24] Montgomery, D. C., Peck, E. A., and Vining, G. G. *Introduction to linear regression analysis*. John Wiley & Sons, 2015.

[25] Mulvenna, M. D., Anand, S. S., and Büchner, A. G. Personalization on the net using Web mining: Introduction. *Commun. ACM 43*, 8 (2000), 122–125.

[26] Nasraoui, O., and Petenes, C. An intelligent Web recommendation engine based on fuzzy approximate reasoning. In *The 12th IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2003, St. Louis, Missouri, USA, 25-28 May 2003* (2003), IEEE, pp. 1116–1121.

[27] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008* (2008), J. T. Wang, Ed., ACM, pp. 1099–1110.

[28] Sparks, E. R., Talwalkar, A., Smith, V., Kottalam, J., Pan, X., Gonzalez, J. E., Franklin, M. J., Jordan, M. I., and Kraska, T. MLI: An API for distributed machine learning. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013* (2013), H. Xiong, G. Karypis, B. M. Thuraisingham, D. J. Cook, and X. Wu, Eds., IEEE Computer Society, pp. 1187–1192.

[29] Sumbaly, R., Kreps, J., and Shah, S. The Big Data ecosystem at LinkedIn. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013* (2013), K. A. Ross, D. Srivastava, and D. Papadias, Eds., ACM, pp. 1125–1134.

[30] Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. S. Presto: Distributed machine learning and graph processing with sparse matrices. In *Eighth Eurosys Conference*

2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013 (2013), Z. Hanzálek, H. Härtig, M. Castro, and M. F. Kaashoek, Eds., ACM, pp. 197–210.

[31] Villari, M., Celesti, A., Fazio, M., and Puliafito, A. Alljoyn lambda: An architecture for the management of smart environments in IoT. In *International Conference on Smart Computing, SMARTCOMP Workshops 2014, Hong Kong, November 5, 2014* (2014), IEEE, pp. 9–14.

[32] Weisberg, S. *Applied linear regression*, vol. 528. John Wiley & Sons, 2005.

[33] Witten, I. H., and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2005.

Appendix A

The AWS-Lambda function code in Python

Listing A.1: The Lambda -function

```
import psycopg2
import boto3
from datetime import datetime

def lambda_handler(event, context):
    print event
    try:
        conn = psycopg2.connect("dbname='postgres' user
            ='' host='lupapistelogdbpsql.cgrw7ecd7cn4.eu
            -west-1.rds.amazonaws.com' password=''")
    except:
        raise RuntimeError("Can not connect to db")

    cur = conn.cursor()
    cur.execute("""SELECT COUNT(applicationId) FROM
        log_data  WHERE applicationId = %s GROUP BY
        applicationId""", (event["applicationId"],))
    count_obj = cur.fetchall();
    if not count_obj or len(count_obj) != 1 or len(
        count_obj[0]) != 1:
        raise RuntimeError("Data base returned
            something unexcepted")
    count = count_obj[0][0]
```

```
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('applications')
item = table.get_item(Key={"id":event["
    applicationId"]})
batch_count = item["Item"]["count"]
count = int(count) + int(batch_count)

submit_time = datetime.strptime(event["submit_time"
    ], '%Y-%m-%d %H:%M:%S.%f')
start_time = datetime.strptime(event["start_time"],
     '%Y-%m-%d %H:%M:%S.%f')
filling_time = (submit_time - start_time).
    total_seconds()
municipality = event["municipality"]
client = boto3.client('machinelearning')
response = client.predict(
     MLModelId='ml-EIvVGmXBdBt',
     Record={
          "municipality" : municipality,
          "action-count" : str(count),
          "filling-time" : str(filling_time),
          "operation": "pientalo"},
     PredictEndpoint='https://realtime.
         machinelearning.eu-west-1.amazonaws.com'
)
return response
```

Appendix B

The Hadoop Pig script

Listing B.1: Pig script to calculated amount of log events per application

```
-- Load log data rows
log_data = LOAD 'log_data' USING PigStorage(';') AS (
    datetime, applicationId, isInfoRequest, operation,
    municipalityId, userId, role, action, target);

-- Select application ids
application_ids = FOREACH log_data GENERATE applicationId;

-- Group rows by application id
grouped_by_id = GROUP application_ids BY applicationId;

-- Calculate rows per application
log_event_counts = FOREACH grouped_by_id GENERATE group as
    id, COUNT(application_ids) AS eventCount;

-- Join original data log_event_counts
joined = join log_event_counts by id, log_data  by
    applicationId;

-- Select features
applications_with_event_counts = foreach joined generate
    applicationId, eventCount, municipalityId;

-- Save results
STORE applications_with_event_counts INTO '/results/grouped
    ' USING PigStorage();
```